PART I

An Introduction to Visual Studio LightSwitch

- **CHAPTER 1:** Prototyping and Rapid Application Development
- **CHAPTER 2:** Getting Started with Visual Studio LightSwitch
- **CHAPTER 3:** Technologies behind a LightSwitch Application
- **CHAPTER 4:** Customizing LightSwitch Applications

1

Prototyping and Rapid Application Development

WHAT YOU WILL LEARN IN THIS CHAPTER

- Coping with the main challenges of line-of-business software development
- Understanding how application prototyping can help you cope with those challenges
- Understanding rapid application development, and how it is related to Visual Studio LightSwitch

Microsoft is known as a company delivering great development tools. To create data-centric applications, for a long time, Microsoft has been offering only two tools that target separate audiences:

- Visual Studio is to be used by a wide range of developers from students and hobbyists, to enterprise developers and architects.
- Microsoft Access (a part of the Office Plus bundle) provides an easy-to-use approach to create data-centric applications for users with very basic development skills.

With Visual Studio, a wide range of applications can be created from the smallest console utilities to highly scalable web applications. The price of this freedom and scalability is that developers must invest a relatively high amount of work to create their applications. Although Visual Studio provides a number of productivity enhancement functions to create data-centric applications, using them requires advanced programming knowledge.

In contrast to Visual Studio, Microsoft Access requires only basic development skills. The simplicity of Access allows users without strong development backgrounds to create

their database tables, forms, and reports. However, the price of this simplicity is that Microsoft Access has strong architecture limitations — it supports only monolith or traditional client-server application architectures. Creating a bit more complex user interface (UI) logic or data validation with Access than the default one requires advanced programming skills.

As a member of the Visual Studio family, Visual Studio LightSwitch is a great new development tool. Microsoft developed this product especially to support rapid application development (RAD) techniques in line-of-business (LOB) application development.

LightSwitch is the golden mean between the simplicity of Access and the flexibility of Visual Studio. With LightSwitch, you can easily create data-centric applications by simply designing data structure and the related UI. To create your own data validation or UI logic requires writing only a few lines of code — and most importantly, you not need to have advanced programming skills. Without any change in your application's structure, you can deploy it either as a desktop application or a scalable web application in the cloud.

When you need to extend an existing LightSwitch application, you can load it into the Professional, Premium, or Ultimate editions of Visual Studio 2010, and extend it with pretty complex business logic, UI behavior, or integrate it with your own back-end systems. Of course, it requires advanced software development knowledge. But you can use the existing LightSwitch application as a springboard, and do not have to create a new one from scratch.

This chapter provides overview about application prototyping and RAD techniques. Here you will learn how these techniques can answer LOB software development challenges, and also understand how Visual Studio LightSwitch does it.

LINE-OF-BUSINESS SOFTWARE DEVELOPMENT CHALLENGES

Today, most companies cannot survive without IT infrastructure supporting their operations. For a long time, *infrastructure* meant only hardware, operating system, and database management systems. Later, other services such as e-mail, collaboration platforms, and systems management services became standard parts of the IT infrastructure. Today, enterprise resource planning (ERP) and customer relationship management (CRM) systems are also part of the IT infrastructure in small and medium businesses.

Although many companies use almost the same IT infrastructure in terms of operating system, database and communication platforms, ERP, CRM, and so on, they still work in different ways with those systems. They all have some unique factors that differentiate them — and their businesses — from competitors on the same market segment. To be unique in this sense, they often need specific software tailored to their requirements and imaginations.

Because of these differences in how businesses use and think about their IT infrastructures, they also need to consider what kind of software to develop to best support their specific business processes. These management applications are often called *line-of-business (LOB) applications*, or *LOB software*.

LOB Software Development

There are many reasons why companies may need to develop LOB software, including the following:

- To create an application that meets business needs not currently met by existing systems
- To develop satellite applications to support existing systems
- To establish an ergonomic user interface (UI) for a legacy system

Traditionally, software development projects involve team members and stakeholders both from the business side and from the IT side. Generally, the business side is responsible for defining the business context and the issues (tasks) to be solved by the LOB application. Also, the business side undertakes managing user acceptance tests — and related quality tests — that validate the solution. The IT side is generally responsible for implementation of the LOB application, including system design, infrastructure, coding, testing, and deployment.

For some activities this division of labor is not so clear-cut. For example, in some companies web design is controlled by business stakeholders, while other companies delegate it to the IT side.

Developing LOB applications is a challenging task. Some of these challenges arise from technical or functional complexity, but the toughest ones reflect the different mindsets of the people involved. In this chapter, you will learn ways to meet many of these challenges.

NOTE *It would be far beyond the scope of one chapter, and indeed one book, to treat all of the LOB application development challenges. This chapter addresses the most significant ones you are likely to experience when working within a LOB application development team — representing either the business side or the IT side.*

Changing Project Environment

The traditional software development life cycle known as the *waterfall model* — whereby the design, implementation, test, and deployment phases follow each other without overlapping — does not work well in today's LOB application projects. Any project that takes more than one day — and most projects (if not all) belong in this category — must meet the challenges of the continuously changing environment surrounding the project. Accordingly, the original requirements, goals, and (at the end of the day) application features change, too. These changes can be legal, political, economic, technological, human, and so on. LOB applications are similarly affected by such changes, because the business environment also undergoes continual, and often rapid, change.

Creating a Requirements Specification

New LOB applications, or functional extensions of existing LOB systems, generally begin their lives with a requirement specification. This document summarizes all functional requirements

(what the system is expected to do) and all quality requirements (performance, service level, UI, robustness, security, and so on), which form the basis for the detailed system specification or system design.

NOTE *Many software development methodologies and frameworks do not use the term* requirement specification*. However, each has some artifact that outlines and describes what the sponsors and users want — whatever that artifact is called. What they share in common is the translation of a "wish list" into a detailed document or prioritized list of required features as agreed upon by members of the project team.*

Creating a clear requirements specification is an integral part of developing a LOB application. If this specification fails to mirror the real-world expectations and uses of the application to be implemented, the result may be a poor or even useless system. In some cases, it may conform to the specification but key users won't like it. Keep in mind that stakeholders from both sides of the aisle (business and IT) generally speak separate languages. Whereas some people quickly grasp a few simple sentences, others process information using screenshots and storyboards, and still others prefer formal descriptions, such as Universal Modeling Language (UML) use cases or activity diagrams.

NOTE Unified Modeling Language *(*UML*) is a general-purpose modeling language that uses a visual model to describe a system. This model is built up from several types of diagrams that define the structure and the behavior of the system. For example, the* use case diagram *describes the functionality (called* use cases *in UML) of the system by means of how users (called* actors *in UML) interact with them. UML was elaborated by James Rumbaugh, Grady Booch, and Ivar Jacobson. It was standardized in 1997 by the Object Management Group (OMG) consortium, and is still managed by this group.*

Very often, a requirements specification is presented to stakeholders as one long document, and the stakeholders must weed through numerous details to find the information they are seeking. These documents typically use the language style of legal contracts, and digesting them is extremely laborious. The best requirements specifications are simple documents, but they can be anything that unambiguously communicates to stakeholders the LOB system to be developed.

NOTE *Not the range, but the content of a requirement specification makes it useful or useless. A good specification describes both* functional requirements *(what functions the system has) and* quality requirements *(how the function should work by means of performance, reliability, user friendliness, and so on). Defining requirements with measurable expectations (". . . this function must retrieve the results in 2 seconds . . .") and using prioritization to separate critical functionality from nice-to-have also adds value to the specification.*

Feedback Frequency

While a LOB application is under development, feedback from key users and business stakeholders about the burgeoning system is critical. If users see the new system only at the very end of the implementation phase, any issues or problems that are found can be time-consuming and expensive to fix, in some cases requiring expenditures that exceed the planned budget. Conversely, if key users want to see the new system's progress every day, that can cause a lot of overhead for development and support.

NOTE *Problems found after the implementation phases of a project often reflect an ambiguous requirements specification and/or false assumptions regarding the application's usability.*

Finding the right balance for feedback frequency is a challenge whose solution will vary according to the project. For some projects, three days or a week might be fine, whereas several weeks might be optimal for others.

For example, while you are in the UI design phase of the project, having two feedback meetings in a week can help you to progress faster. Later, when you are about to elaborate specific business modules, having a review meeting every two weeks could be enough.

NOTE *Don't underestimate the importance of finding the right feedback frequency, which can be a lifesaver. Especially for long and complex projects, using Visual Studio LightSwitch can significantly help you to communicate your ideas and understanding of LOB application requirements — as you will see after completing Part I of this book.*

You can even vary how often you provide feedback to your users. At the beginning (during the conception phase or while designing the application), it might be appropriate to communicate progress every few days. In some cases, you can even carry out a feedback cycle within one day.

For example, in a morning meeting, you might ask key users for feedback about a new screen issued the previous day. In the afternoon, you could present how you plan to address that feedback. Later, after implementing the desired feature(s), you could ease the initial frequency. When you are about to prepare for a pilot deployment or the production deployment, feedback frequency should again be increased.

APPLICATION PROTOTYPING

There are many ways to manage the challenges mentioned previously and mitigate the risks associated with them. The goal is to prevent risks that result from wrong information or insufficient information. For example, if an order management process is not entirely clear because you do not know the CRM system that stores customer information, the lack of this information is a risk. Similarly, not knowing all the attributes that should be entered for a new order is also a risk.

Application prototyping is a tool for managing such situations — and many challenges related to the human factor — as well as mitigating associated risks. While it is not the only tool, it is one of the best.

Prototyping, and the resulting application prototype, is defined in various ways. The essence of prototyping is the creation of a functional, and perhaps somewhat limited, model of the final application. Unlike specification and design documents that use literal or formal descriptions, this working model can be readily understood by key users.

For example, try to explain a UML user activity diagram to key users! Even if they understand it, they cannot truly appreciate how it will be implemented. Conversely, if you create a prototype of the activity, such as an order process, using a storyboard that represents the same UML diagram, users will have a greater level of confidence that the final product will be the right one.

You can also use application prototyping to obtain required information from users in an indirect way. If users are unable to clearly explain exactly what they want — which is not uncommon — you can create a prototype that implements an incomplete, or even obviously inferior, model. When you present such a prototype to key users, they can usually tell you what's wrong with it immediately, or what's missing.

The rest of this section describes the various kinds of prototypes you can use. Depending on your goals — that is, what you want to communicate — you might use one or more on a single LOB project.

Wireframe Models

The term *wireframe* has been used in three-dimensional (3D) modeling for a long time, especially in 3D computer graphics. This term is also used to describe UI prototypes, mainly for presenting website illustrations. A "wireframe" in this context depicts the layout of the fundamental elements in the user interface. Figure 1-1 shows an example of a wireframe describing the home page of a fictional company.

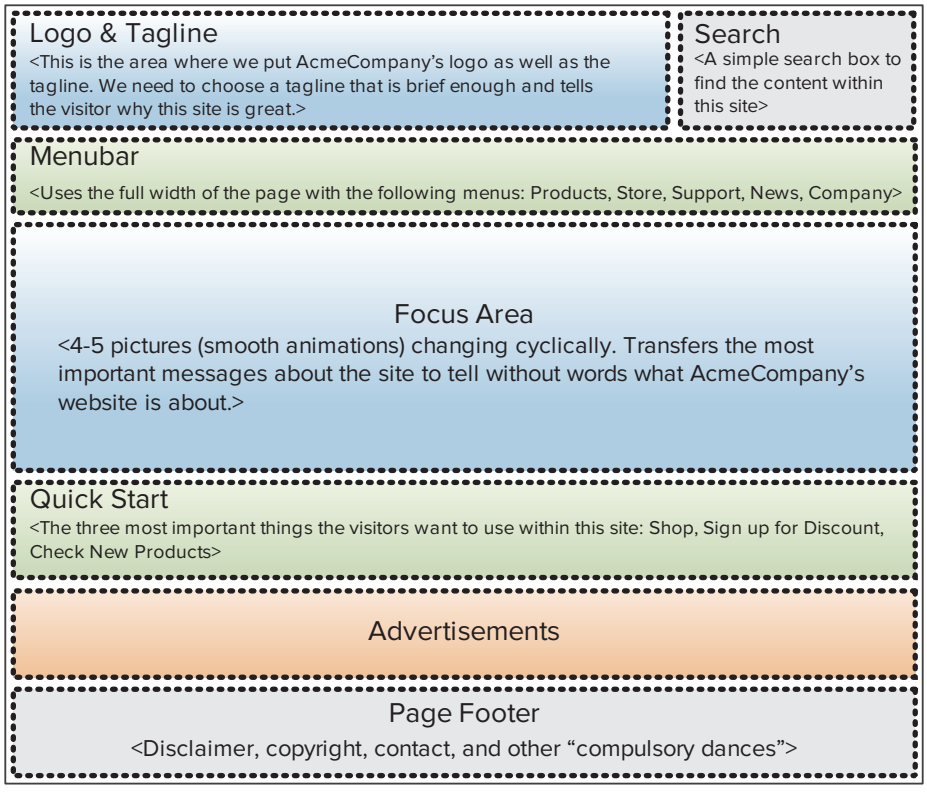

FIGURE 1-1: A wireframe example describing a home page

Note the simplicity of this wireframe. It does not contain a high-level, sophisticated design because its aim is not to present the graphical look of the home page but rather to enable key users to focus on its structure and elements. The colors used in this wireframe are just for separating layout segments visually; they are not the real colors to be used in the final design.

You may be wondering why it is useful to create a wireframe instead of a model that more closely resembles the final state of the home page. Wireframe models offer a few advantages over more detailed prototypes, including the following:

- They are relatively cheap to create. Even a whiteboard can be used to create them. Using wireframes, you can save time and the expense of creating possible unsuitable UI models.
- You can give a wireframe to key users and they will quickly have a basic understanding of your intention. If something is wrong, it can be corrected instantly.
- When you present a graphically designed home page prototype to users, their attention is focused on the *style* of the page — the logo used, the font type, and so on — instead of the *structure* of the page. Of course, later you must present them with the graphical design. But first the structure should be grabbed.

- A wireframe is also a good start for the final web design, because it relays a lot of information about the intentions of key users that is useful to the experts who create the graphical design.

Proof-of-Concept Models

For any project, any piece of information you are lacking is a source of risk. If you are unsure how to carry out any of the tasks in your to-do list, then that is also a source of risk. In situations where you know *what* you are expected to do, but not *how* you do it, you need to do some research.

A great method for performing this research is to use a prototyping technique called *proof-of-concept modeling*. Instead of thinking and making plans about how to solve a specific issue, you build a very simplified working model to prove the feasibility of your idea. If that is usable, you can use this model later, of course, with the necessary modifications. This model is called "proof-of-concept" because it can either confirm that your hypothetical solution works or disprove its viability.

NOTE *Let's assume you need to implement rapid search functionality for customers. Instead of the traditional approach whereby users type a part of the customer name and click a button to retrieve a list of matching customers, you need the capability to repeatedly reduce the number of matching names as users enter additional letters in the search box. You may assume you can do this but you cannot be completely sure. However, building a simple proof-of-concept model may help you. This precludes guessing whether you can meet this challenge, and it provides the information you need to plan how you'll implement the final function.*

Low-Fidelity Prototypes

In some situations, you cannot avoid implementing parts of the growing application in order to communicate how they work and what they do. Wireframes can indicate the layout of a particular UI, and proof-of-concept models implement a very simple (and probably only technical) aspect of the same UI.

If key users need more information to understand the solution you plan to provide them, you may need to create working prototypes that can be used to demonstrate and test your ideas. These working parts are not simple models, but real applications. In many cases, you can create a *low-fidelity prototype* that is sketchy and incomplete but represents the main characteristics of the target function.

For example, suppose you are required to demonstrate an order entry function implemented as a four-step process emulating the access to several back-end systems. You can create a low-fidelity prototype that leads the user through this process. Instead of just modeling the workflow, the prototype really implements it, but it omits parts that retrieve data from the CRM at the back end and write orders back in the ERP.

Because this prototype focuses on the process, you can create only a sketchy UI with a very basic design and a draft layout. Moreover, it does not have to deal with authentication, business logic parameterization, or any other things that are not closely related to the workflow.

High-Fidelity Prototypes

In some situations, you must create a *high-fidelity prototype* that provides much more detail about the intended functionality, and serves as evidence that the functionality can be carried out in the outlined way.

Returning to the low-fidelity order entry example, you may be asked to demonstrate this function in more detail. In this case, your prototype should not only mimic accessing the back-end systems, but also use them to retrieve and check customers from the CRM, and return the entered order to the queue of the ERP system. In addition, the sales department would like to ensure that the UI of the workflow is intuitive and provides a great user experience.

In this case, you could implement a high-fidelity prototype that is very close to the final solution.

NOTE *With the help of Visual Studio LightSwitch, you can create both low-fidelity and high-fidelity models easily.*

RAPID APPLICATION DEVELOPMENT

Rapid application development (RAD) is a software development methodology that uses minimal planning and rapid prototyping, rather than thorough application design and waterfall-like models. The planning of the software using the RAD approach is generally interleaved with the coding — or implementation — phase of the software. This approach is very useful, because it enables software to be developed much faster — and makes it easier to accommodate the continuously changing project environment than waterfall-like models do.

The term "rapid application development" was introduced in 1991 by James Martin, who used it to describe a software development process that emphasizes an iterative approach to the whole construction phase, and handles prototypes as first-class citizens of the implementation process.

RAD is not a single, particular software development methodology. Rather, it is a generic name for concrete methodologies that primarily rely on iterations and prototypes in contrast to the traditional waterfall methodologies. RAD has many flavors, including generic agile software development methods, as well as Scrum, Extreme Programming (XP), Lean Software Development (LD), or Joint Application Development (JAD).

NOTE *This book does not cover individual software development methodologies, so if you want more details about these RAD technologies, use your search engine of choice to search online for more information.*

The main strength of the RAD approach is that you can avoid a vast amount of rework in your software development projects. Rework most often occurs in the following two cases:

- You implement a piece of software in a wrong way, or with poor quality. In this case, you must spend resources to fix development issues.
- You implement a wrong piece of software — not the one expected by its key users, but something else. In this case, you must recreate the particular piece from the beginning.

Of course, RAD and other agile software development methodologies do not prevent you from making poor quality (or buggy) software. However, they can help you mitigate the risk of constructing a wrong piece of software. By building a prototype, you can verify that you are building the right functionality according to the right quality expectations — in other words, the product your key users want. While some rework may be required when you need to prepare a new prototype to replace a faulty one, this rework still costs less than recreating from scratch a software module that is intended to be a final product.

In some situations the RAD approach of making prototypes does not add much value to your development process. When your specification is very detailed, and you do not have significant technology risks (because you can handle them routinely), you can start implementing final products instead of prototypes.

RAD Tools

Today, practically all development tools and environments support the RAD approach. All tools promise to provide functions that help you to be agile and productive. Some of them add new visual design features to enhance manual code writing. Others use code libraries that dramatically reduce the length of source code. Several tools use wizards that lead you through a complex process. As technology evolves, developers expect increasingly sophisticated features from a RAD tool. While today the expectations are very high, this has not always been the case.

The following sections look at a few tools that are good examples of how RAD was implemented a few years ago.

Visual Basic

Visual Basic 1.0 (released in May 1991) was the first RAD tool for the Windows platform. For a long time, Windows development was a field on which only C and C++ programmers could play. The smallest "Hello, World" program for Windows was about 100 lines of code, whereas the statements to actually print out the "Hello, World" text required only about a dozen lines.

Visual Basic 1.0 took the development community by storm, and totally changed the programming model from code-oriented development to UI-oriented development. Whereas C and C++ programmers used resource files to describe the UI, Visual Basic invented the concepts of forms, controls, and visual GUI construction. The reusability and extensibility of forms and controls was a main design goal in Visual Basic. Developers could create their own custom controls using generic — or, conversely, application specific — properties and methods.

From a developer's point of view, Visual Basic was a real RAD tool. Developers could drag components and controls from a toolbox onto the surface of forms, and place them into the desired

position. The behavior of controls could be changed by setting up the properties of visual elements. Forms and their controls had events represented by methods, with which programmers could code the logic of the application.

Visual Basic has evolved a lot since then, but using current development tools (such as Visual Studio), you can recognize that the elements of the integrated development environment (IDE) still resemble those used in the old versions. Figure 1-2 shows a screenshot from an old Visual Basic version running under Windows 95.

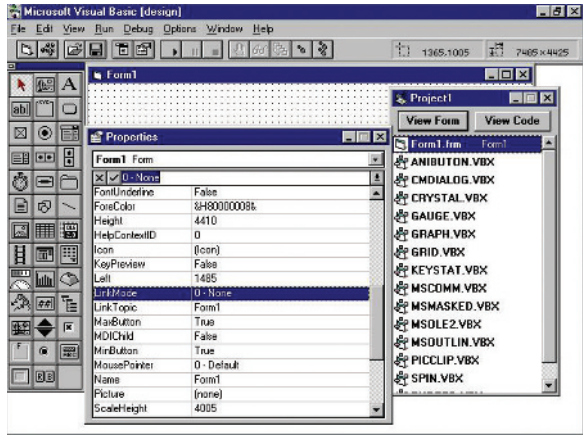

FIGURE 1-2: An old Visual Basic IDE running under Windows 95

Microsoft Access

Microsoft Access 1.0 was released in November 1992. It offered a relational database management system (RDBMS) for desktop applications — combined with a graphical user interface (GUI) and great visual tools. Developers could easily design database schemas, enter data with simple forms, and create reports. They could use the Visual Basic programming language (VBA, which was included with the product) to add code and create real applications for end-users.

Figure 1-3 shows the database window of the Northwind sample application in Microsoft Access 97.

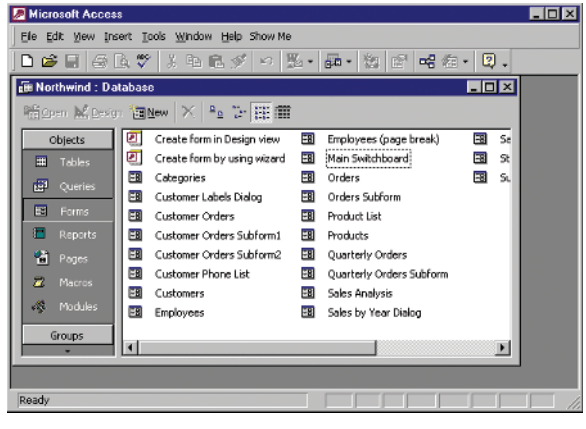

FIGURE 1-3: Microsoft Access form objects in the design environment

Originally, Access used its own database engine — called Microsoft Jet Database Engine — but after the release of version 2.0, it could use external database tables. Microsoft Access 2000 allowed developers to work directly with SQL Server databases.

Access was — and still is — a great RAD tool. It lowered the entry barrier to relational database programming. While creating applications for most RDBMSs required a set of applications using several tools and programming languages, with Access it was much simpler and quicker.

Delphi

Delphi was originally developed by Borland, and its first version was released in 1995. This tool had an IDE that was very similar to the Visual Basic IDE. Delphi was designed to be a RAD tool that supported developing database applications, including simple ones and even enterprise applications.

It used the Object Pascal language — a successor of Turbo Pascal — which provided full object-oriented programming (OOP) capabilities, in contrast to Visual Basic.

The product evolved very fast with five versions released in the first five years of its life. Delphi was the first RAD tool capable of compiling 32-bit applications for Windows. It became very popular among enterprise developers because of its RAD features. It provided more than 100 components (elements of the Delphi Visual Component Library) that developers could immediately drop onto the Designer surface. In addition, developers could easily create their own visual components, and add them to the existing library.

NOTE *At the time, Visual Basic also provided a separate control development SDK that made it possible to create additional components called* custom controls*. However, Delphi offered a very intuitive and much faster way to develop controls, because the IDE was designed with component reusability in mind.*

Figure 1-4 shows the Delphi 7 IDE. The largest part of the toolbar at the top of the IDE contains component category tabs and components.

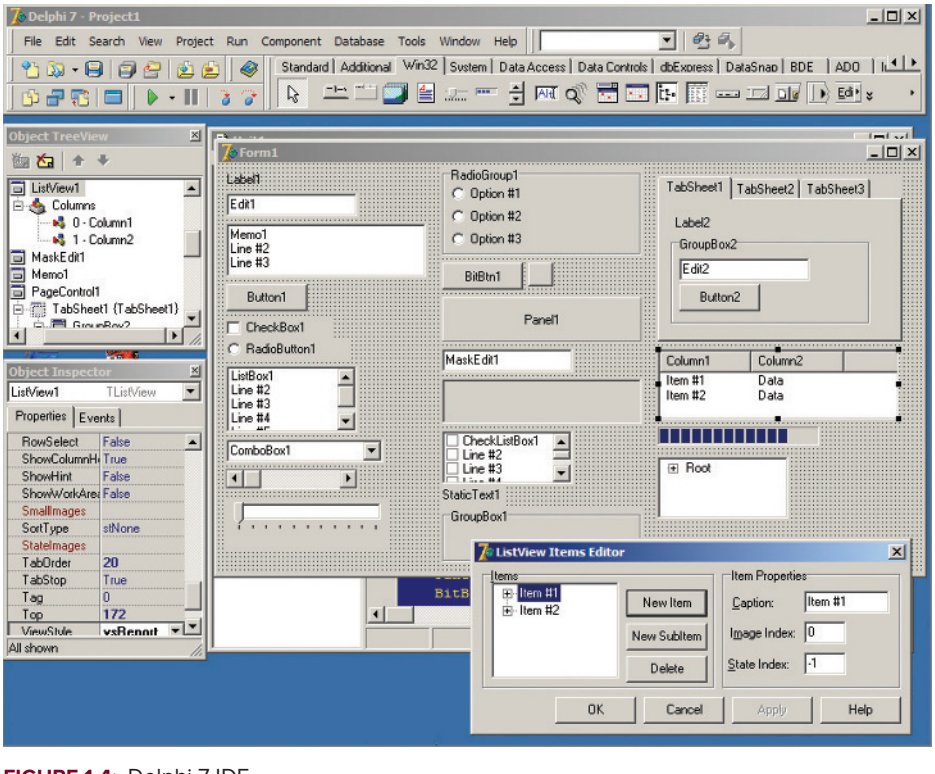

FIGURE 1-4: Delphi 7 IDE

Visual Studio LightSwitch and RAD

Visual Studio LightSwitch is the newest member of the Visual Studio family. Like Visual Basic, Microsoft Access, and Delphi, it is a RAD tool; but as its name suggests, it aims to make the development of LOB applications as easy as flipping a switch.

Most RAD tools are development environments made for programmers, and they provide productivity tools that enable the faster creation of applications. Visual Basic and Delphi are definitely such tools. Microsoft Access is a bit different in that it is not a generic development tool, but rather is intended for creating database applications for the desktop.

Visual Studio LightSwitch is a RAD tool that makes data-centric LOB application development available not only for developers, but also for business analysts, consultants, and IT experts working on business projects. As suggested earlier, "LightSwitch" in the name of the new product symbolizes how easy it is to create LOB applications.

In contrast to RAD tools like Visual Studio or Delphi, LightSwitch is designed to support prototyping with a minimal amount of coding — or no coding at all. Compared to the data-centric RAD style offered by Microsoft Access, which supports the traditional client-server separation of application layers, LightSwitch provides a clean and very sophisticated three-tier application architecture, and takes care of all the plumbing that binds the layers into a working application. With Visual Studio LightSwitch, you can create complete LOB applications, as well as low-fidelity or high-fidelity application prototypes.

In the following three chapters, you will learn about the fundamentals of LightSwitch and get a taste of this great tool. Chapter 2 focuses on the first steps to getting started with the product, while Chapter 3 treats the key technologies behind LightSwitch. You have several ways to customize your prototypes built with LightSwitch, as Chapter 4 demonstrates.

SUMMARY

Writing LOB applications has many challenges — mostly related to specifications and communication among the members of the development team. Projects that cannot meet these challenges can easily fail.

One of the main development challenges is addressing the often unclear or unspoken expectations of key users. Key users can rarely tell you their exact expectations, either because they are unable to communicate exactly what kind of functionality they expect or they are unsure how to use a certain business function.

Prototyping (that is, developing a working model to test ideas and feasibility) is a great technique to overcome these situations. Depending on your particular scenario, you can use several kinds of prototypes, including wireframes, low-fidelity and high-fidelity prototypes, or proof-of-concept models, to bridge the sea of missing information or clear communication between key users and the software construction team.

Most RAD tools — including Visual Studio LightSwitch — have robust features to support you in the creation of communicable prototypes.

In Chapter 2, you'll extend your knowledge about the role of LightSwitch among the members of the Visual Studio family. You will learn both how to install LightSwitch and how to create your first application — without writing any lines of code.

EXERCISES

1. Enumerate and explain a few challenges of LOB application development with regard to the human factor.
2. Explain what a wireframe is.
3. Explain what a proof-of-concept model is.
4. What is the fundamental principle of rapid application development (RAD)?
5. List a few methodologies that are based on the RAD principle.

NOTE *Answers to the Exercises can be found in the Appendix.*

WHAT YOU LEARNED IN THIS CHAPTER

TOPIC	KEY CONCEPTS
Line-of-business (LOB) application	A LOB application participates in managing the business processes of an organization.
Challenge of changing project environment	The environment of a project (one with the duration of several weeks, months, or even years) changes. Project planning must be undertaken with possible changes (legal, political, economic, human, technological, and so on) kept in mind.
Challenge of feedback frequency	Both developers and key users need feedback from each other during a LOB development project to mutually confirm that the right functionality is about to be implemented in the right way. Finding the optimal frequency for this feedback is essential to the project's success.
Wireframe	A wireframe is a prototype that depicts the layout of the fundamental elements in the user interface (UI). It emphasizes structure over graphical design.
Proof-of-concept model	A proof-of-concept model is a working prototype that can be used to check the feasibility of an idea. It focuses on the technical details to be checked, without implementing other application details.
Rapid application development (RAD)	Rapid application development (RAD) is a software development methodology that uses minimal planning and rapid prototyping, rather than thorough application design and waterfall-like models.